http://www4.ncsu.edu/~dkim2/csc548/csc548.htm

Revised on Nov 21, 2009

CSC548

Homework 5

Donghoon Kim
dkim2@ncsu.edu

1. The progress of my project :

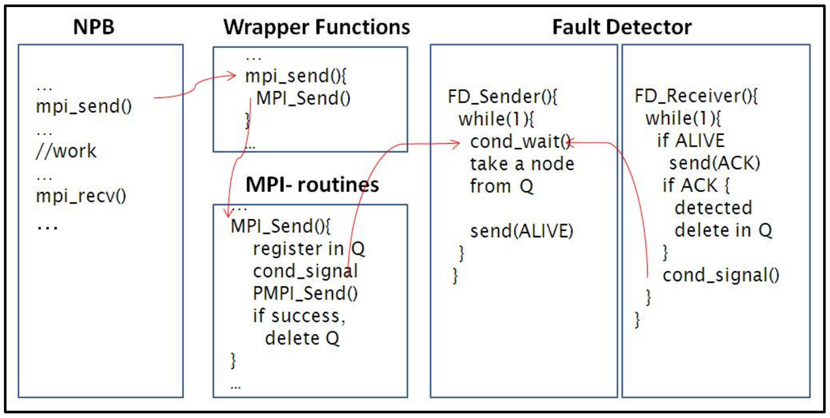

Figure 1. The Diagram of Fault Detector

1.1 Implementation

Figure 1 represents the diagram of the fault detector for sporadic communication which I have implemented so far. Suppose that the MPI application such as NAS Parallel Benchmark (NPB) runs on Massive parallel processing (MPP) environment. As the most MPI tools utilize, MPI profiling layer (PMPI) intercepts MPI calls during application execution. When MPI_Init is called in application, Fault Detector (FD) is also launched since the code for calling FD is inserted in MPI_Init. On running FD, FD executes its own separate routine independent from the application. The new communicator, FD_COMM is used for FD communication routines to provide independent execution among FDs at each node. FD consists of two threads, FD sender and FD receiver. The Wrapper Function (WF) is for the application written in Fortran at which WF links MPI functions of the application to PMPI. Whenever MPI communication routines (e.g. MPI_Send or MPI_Isend) in the application program are called, the corresponding PMPI routines are also executed. Each PMPI routine has three steps such as pre-processing, PMPI function call, post-processing. As an example of MPI_Send, Pre-processing registers a destination node with current time in Queue and sends a signal to FD sender in case of waiting for a signal since Queue is empty. PMPI function call executes normal function like PMPI_Send for the application execution. Post-processing deletes the destination node registered in Queue if returned SUCCESS in PMPI_Send. FD manages the status of neighbor nodes with Queue which is implemented using a doubly liked list with node ID, timestamp, check-in and so on. The following is the more detailed description of both FD sender and FD receiver.

- FD sender : It is supposed to send the ALIVE message unless Queue is empty. Before sending ALIVE message, FD sender checks the timestamp of a destination node in Queue whether the delay time passes the timestamp or not. The purpose of the delay time is not to make a redundant message. If delay time passes the timestamp, FD sender sends ALIVE message to that destination node and then flips check-in flag to indicate that FD already sent ALIVE message and is waiting for ACK message from that node with updated timestamp. When FD rechecks this node for the next turn, FD sender is able to suspect this node as a node failure if this node still exists in Queue. FD sender sorts Queue by updated timestamps in ascending order after one cycle.

- FD receiver : It is supposed to receive either ALIVE or ACK message. FD receiver probes periodically whether a message is arrived or not. On receiving a message, FD receiver takes the next action according to MPI_TAG. FD receiver reply back sends back ACK message for ALIVE message while deleting a node ID from Queue for ACK message.

1.2 Test

FD has been tested with the CG in NAS Parallel Benchmark suite as a basic test. The CG benchmark is written in Fortran so that WF is called whenever MPI routine executes. Fortran compilers are different, that is, one of the following function name is used for MPI_Send as an example :

- mpi_send_
- mpi_send__
- MPI_SEND
- MPI_Send

*mpi_send_* is used on opt10. The all arguments are pointer arguments in WF. Furthermore, the *ierr* argument at the end of the argument list in Fortran is not used in C because the ierr is an integer and has the same meaning as the return value of the routine in C.

2. Open problem

One of open problems is abnormal termination. FD terminates abnormally with MPI error message when an application program calls MPI_Finalize. It is because of the following reasons. First, both FD sender and FD receiver are still running while the application is terminated. Second, although identical MPI application runs on one or more machines, MPI_Finalize could be called at different time according to various performance at each machine which would cause awkward communication execution on FD. Third, PMPI is not able to call PMPI_Finalize because blocking operations are still running so that this routine may block until the message is arrived or sent. The solution of abnormal termination is to check how blocking operation works and synchronization. When MPI_Finalize is called, PMPI wrapper for MPI_Finalize calls pthread_cancel for both FD sender and FD receiver and calls MPI_Barrier for FD synchronization and then executes PMPI_Finalize.

3. Remaining work

I should implement more PMPI wrappers of MPI communication routines according to NAS Parallel Benchmark suite. After done MPI communication routines, a global view for failure nodes should be added so that each node shares the list of failed nodes and is aware of failed nodes. Refer to the timeline.

4. Timeline :

Timeline for the remainder of the project	
~~11/2~~	~~complete basic FD~~
~~11/9~~	~~add other communication routines in NPB benchmark.~~
11/16	debugging & add a global view for failure nodes
11/23	Performance test
11/30	Write a report